Lesson 7

• From DSA to Regular Expression
• From Regular Expressions to DSA, directly
Motivations: yesterday’s exercise 7(b)

• Write a regular expression over the set of symbols \{0,1\} that describes the language of all strings having an even number of 0’s and of 1’s
  – Not easy….
  – A solution: \((00|11)*((01|10)(00|11)*(01|10)(00|11)*)^*\)
  – How can we get it?

• Towards the solution: a deterministic automaton accepting the language
• But how do we get the regular expression defining the language accepted by the automaton?
Regular expressions, Automata, and all that...
From automata to Regular Expressions

• Three approaches:
  – Dynamic Programming [Scott, Section 2.4 on CD] [Hopcroft, Motwani, Ullman, *Introduction to Automata Theory, Languages and Computation*, Section 3.2.1]
  – Incremental state elimination [HMU, Section 3.2.2]
  – Regular Expression as fixed-point of a continuous function on languages
DFAs and Right-linear Grammars

• In a right-linear (regular) grammar each production is of the form \( A \rightarrow wB \) or \( A \rightarrow w \) (\( w \in T^* \))
• From a DFA to a right-linear grammar

![Diagram of DFA and corresponding right-linear grammar]

A \rightarrow \epsilon | 1B | 0D
B \rightarrow 1A | 0C
C \rightarrow 0B | 1D
D \rightarrow 0A | 1C

• The construction also works for NFA
• A similar construction can transform any right-linear grammar into an NFA (productions might need to be transformed introducing new non-terminals)
Kleene fixed-point theorem

• A complete partial order (CPO) is a partial order with a least element $\bot$ and such that every increasing chain has a supremum

• Theorem: Every continuous function $F$ over a complete partial order (CPO) has a least fixed-point, which is the supremum of chain

$$F(\bot) \leq F(F((\bot))) \leq \ldots \leq F^n(\bot) \leq \ldots$$
Context Free grammars as functions on the CPO of languages

• Languages over $\Sigma$ form a *complete partial order* under set inclusion

• A context free grammar defines a continuous function over (tuples of) languages
  
  \[ A \rightarrow a \mid bA \quad F(L) = \{a\} \cup \{bw \mid w \in L\} \]

• The language generated by the grammar is the least-fixed point of the associated function
  
  \[ \emptyset \subset \{a\} \subset \{a, ba\} \subset \{a, ba, bba\} \subset \ldots \subset \{b^n a \mid n \geq 0\} \]

• In the case of right-linear grammars we can describe the least fixed-point as a regular expression
  
  \[ \text{Lang}(A) = b^*a \]
Example: from right-linear grammar to regular expression

1) Substitute D in A and C
   \[ A \rightarrow \varepsilon | 1B | 0(0A | 1C) \]
   \[ B \rightarrow 1A | 0C \]
   \[ C \rightarrow 0B | 1D \]

2) Substitute B in A and C
   \[ A \rightarrow \varepsilon | 1(1A | 0C) | 0(0A | 1C) \]
   \[ B \rightarrow 1A | 0C \]
   \[ C \rightarrow 0(1A | 0C) | 1(0A | 1C) \]

3) Put C in form C = \( \alpha | \beta C \)
   \[ A \rightarrow \varepsilon | 1(1A | 0C) | 0(0A | 1C) \]
   \[ B \rightarrow 1A | 0C \]
   \[ C \rightarrow 01A | 10A | (00 | 11)C \]

4) Solve C: \( C = (00 | 11)^*(01A | 10A) \)

5) Factorize C in A
   \[ A \rightarrow \varepsilon | 11A | 00A | (10 | 01)C \]

6) Substitute C in A
   \[ A \rightarrow \varepsilon | 11A | 00A | (10 | 01)(00 | 11)^*(01A | 10A) \]

7) Put A in form A = \( \alpha | \beta A \)
   \[ A \rightarrow \varepsilon | (11 | 00 | (10 | 01)(00 | 11)^*(01 | 10))A \]

8) Solve A: \( A = (11 | 00 | (10 | 01)(00 | 11)^*(01 | 10))^* \)

The other solution: \( (00|11)^*((01|10)(00|11)^*(01|10)(00|11))^* \)
Regular expressions, Automata, and all that…

- Regular Expressions
- Thompson algorithm
- Non-Deterministic Finite Automata
  - Subset construction
  - Least fixed-point of function on languages
- Right-linear (Regular) Grammars
  - Directly!
  - Easy!
- Deterministic Finite Automata
  - Minimization (Partition/Refinement)
From Regular Expression to DFA Directly

• The “important states” of an NFA are those with a non-ε outgoing transition,
  - if \( \text{move}(\{s\}, a) \neq \emptyset \) for some \( a \) then \( s \) is an important state

• The subset construction algorithm uses only the important states when it determines \( \varepsilon\text{-closure}(\text{move}(T, a)) \)
What are the “important states” in the NFA built from Regular Expression?

\[ r_1 | r_2 \]

\[ r_1 r_2 \]

\[ r^* \]
From Regular Expression to DFA Directly (Algorithm)

• The only accepting state (via the Thompson algorithm) is not important

• Augment the regular expression \( r \) with a special end symbol \# to make accepting states important: the new expression is \( r\# \)

• Construct a syntax tree for \( r\# \)

• Attach a unique integer to each node not labeled by \( \varepsilon \)
From Regular Expression to DFA Directly: Syntax Tree of  \((a|b)^*abb\#\)
From Regular Expression to DFA Directly: Annotating the Tree

- Traverse the tree to construct functions $\text{nullable}$, $\text{firstpos}$, $\text{lastpos}$, and $\text{followpos}$
- For a node $n$, let $L(n)$ be the language generated by the subtree with root $n$
- $\text{nullable}(n)$: $L(n)$ contains the empty string $\varepsilon$
- $\text{firstpos}(n)$: set of positions under $n$ that can match the first symbol of a string in $L(n)$
- $\text{lastpos}(n)$: the set of positions under $n$ that can match the last symbol of a string in $L(n)$
- $\text{followpos}(i)$: the set of positions that can follow position $i$ in any generated string
From Regular Expression to DFA Directly: Annotating the Tree

<table>
<thead>
<tr>
<th>Node $n$</th>
<th>$\text{nullable}(n)$</th>
<th>$\text{firstpos}(n)$</th>
<th>$\text{lastpos}(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leaf $\epsilon$</td>
<td>true</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>Leaf $i$</td>
<td>false</td>
<td>${i}$</td>
<td>${i}$</td>
</tr>
<tr>
<td>$c_1$/$c_2$</td>
<td>$\text{nullable}(c_1)$ or $\text{nullable}(c_2)$</td>
<td>$\text{firstpos}(c_1)$ $\cup$ $\text{firstpos}(c_2)$</td>
<td>$\text{lastpos}(c_1)$ $\cup$ $\text{lastpos}(c_2)$</td>
</tr>
<tr>
<td>$c_1$.$c_2$</td>
<td>$\text{nullable}(c_1)$ and $\text{nullable}(c_2)$</td>
<td>$\text{if}$ $\text{nullable}(c_1)$ $\text{then}$ $\text{firstpos}(c_1) \cup \text{firstpos}(c_2)$ $\text{else}$ $\text{firstpos}(c_1)$</td>
<td>$\text{if}$ $\text{nullable}(c_2)$ $\text{then}$ $\text{lastpos}(c_1) \cup \text{lastpos}(c_2)$ $\text{else}$ $\text{lastpos}(c_2)$</td>
</tr>
<tr>
<td>$c_1$</td>
<td>true</td>
<td>$\text{firstpos}(c_1)$</td>
<td>$\text{lastpos}(c_1)$</td>
</tr>
</tbody>
</table>
From Regular Expression to DFA
Annotating the Syntax Tree of $((a|b)^*abb)#$

```
nullable

{1,2} * {1,2} {1,2}
 {1,2} {1,2}
 {1} a {1}  {2} b {2}

{1} a {1}  {2} b {2}
```

```
{1,2} * {1,2} {3} {3}
{1,2,3} {3} {4} b {4}
{1,2,3} {4} {5}
{1,2,3} {5}
```

```
{6} # {6}
```

```
firstpos

lastpos
```
From Regular Expression to DFA

followpos on the Syntax Tree of \((a|b)^*abb\#\)
From Regular Expression to DFA
Directly: *followpos*

\[
\text{for each node } n \text{ in the tree do}
\]
\[
\text{if } n \text{ is a cat-node with left child } c_1 \text{ and right child } c_2 \text{ then}
\]
\[
\text{for each } i \text{ in } \text{lastpos}(c_1) \text{ do}
\]
\[
\text{followpos}(i) := \text{followpos}(i) \cup \text{firstpos}(c_2)
\]
\[
\text{end do}
\]
\[
\text{else if } n \text{ is a star-node}
\]
\[
\text{for each } i \text{ in } \text{lastpos}(n) \text{ do}
\]
\[
\text{followpos}(i) := \text{followpos}(i) \cup \text{firstpos}(n)
\]
\[
\text{end do}
\]
\[
\text{end if}
\]
\[
\text{end do}
\]
From Regular Expression to DFA Directly: Example

<table>
<thead>
<tr>
<th>Node</th>
<th>followpos</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>{1, 2, 3}</td>
</tr>
<tr>
<td>b</td>
<td>{1, 2, 3}</td>
</tr>
<tr>
<td>a</td>
<td>{4}</td>
</tr>
<tr>
<td>b</td>
<td>{5}</td>
</tr>
<tr>
<td>b</td>
<td>{6}</td>
</tr>
<tr>
<td>#</td>
<td>-</td>
</tr>
</tbody>
</table>
From Regular Expression to DFA
Directly: The Algorithm

\[ s_0 := \text{firstpos}(\text{root}) \] where \text{root} is the root of the syntax tree for \((r)\#
\]

\[ \text{Dstates} := \{s_0\} \] and is unmarked

\textbf{while} there is an unmarked state \(T\) in \(\text{Dstates}\) \textbf{do}

\quad mark \(T\)

\quad \textbf{for} each input symbol \(a \in \Sigma\) \textbf{do}

\quad \quad \text{let } U \text{ be the union of } \text{followpos}(p) \text{ for all positions } p \text{ in } T
\quad \quad \text{such that the symbol at position } p \text{ is } a

\quad \quad \textbf{if } U \text{ is not empty and not in } \text{Dstates} \textbf{then}

\quad \quad \quad add \(U\) as an unmarked state to \(\text{Dstates}\)

\quad \quad \textbf{end if}

\quad D\text{tran}[T, a] := U

\quad \textbf{end do}

\textbf{end do}